Distributed Systems Notes

Sam Kirk

July 2024

Week 1

Acronym Guide

- **RMI** - Remote Method Invocation
- **IPC** - Inter-Proccess Communication
- **MPI** - Message Passing Interface
- **HPC** - High performance computing
- **TCP** - Transmission Control Protocol

Week 2

Java: Write once, run everywhere

Test driven development

JUnit is a test framework. Makes test driven development easy. Add test methods directly into code.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import example.util.Calculator;
import org.junit.jupiter.api.Test;

class MyFirstTests {
    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }
}
```

0.1 Serialisation

Any object moving between vitual machines must be serialisable. So when sending bytes along the network the class needs to be packed up and sent as zeros and ones. User defined classes are not serialisable by default. Can be done using JSON, XML and Protocol Buffers (Google).

Threads

Concurrency within the one process. No need to switch memory spaces so its much faster and locality is preserved. Data is shared between threads of execution No costly IPC (Inter-process Communication), but it does require coordination. So there is not isolation between threads which means one thread can corrupt entire process.

See lecture 2, 1:37 for simple thread code example.

Assignment 1

We are building a client-server model, where the the calculator server provides computation services, and the client requests these services.

RMI is a Java API that allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM.

An RMI Registery is a simple naming service that allows clients to obtain a reference to a remote object. It acts as a directory for locating remote objects.

Remote Interface

The remote interface (Calculator.java) defines the methods that can be called remotely. It extends java.rmi.Remote.

Remote Object Implementation

The class implementing the remote interface (CalculatorImplementation.java) provides the actual implementation of the methods defined in the remote interface. This class should extend UnicastRemoteObject and implement the remote interface.

Server Program

The server program (CalculatorServer.java) creates an instance of the remote object implementation, binds it to the RMI registry, and waits for client requests.

Client Program

The client program (CalculatorClient.java) looks up the remote object in the RMI registry and invokes methods on it.

Stub

Think of a stub as a "helper" on the client side. When a client wants to call a method on a remote object (an object that exists on a different computer), the client actually talks to the stub instead of the remote object directly. The stub acts like a placeholder for the real object. It takes the client's request, packages it up, sends it over the network to the server where the real object lives, and then sends the response back to the client. So, the stub "stands in" for the remote object and makes it seem like the object is local to the client.

Proxy

A proxy is a broader term that means something similar to a stub but can be used in different contexts, not just in RMI. In general, a proxy is an object that controls access to another object. In the case of RMI, the stub acts as a proxy for the remote object because it controls access to it, managing the network communication between the client and the server.

Key Java Concepts

Interface: In Java, an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. The methods in interfaces are abstract, meaning they don't have a body and must be implemented by the classes that implement the interface.

Extends: The extends keyword is used to indicate that a class is inheriting from another class. In Java RMI, interfaces extend the Remote interface to indicate that they can be called remotely.

Implements: A class uses the implements keyword to implement an interface. This means the class provides concrete implementations for the methods defined in the interface.

Throws: In Java, the throws keyword is used in method signatures to indicate that the method may throw certain exceptions. For example, remote methods in RMI must declare that they can throw RemoteException, which indicates a communication-related exception that can occur during a remote method call.

RemoteException: A checked exception that indicates an issue during a remote method call, such as network problems. Methods in a remote interface must declare this exception.

File Explanations

Calculator.java: This is the remote interface that defines the methods the client can call remotely. It extends Remote and all methods throw RemoteException.

CalculatorImplementation.java: This class implements the Calculator interface. It extends UnicastRemoteObject to make it a remote object. The constructor must throw RemoteException, and all methods must provide implementations for the interface methods.

CalculatorServer.java: This is the server program. It creates an instance of the remote object implementation, registers it with the RMI registry under the name "CalculatorService," and waits for client requests.

CalculatorClient.java: The client program locates the RMI registry, looks up the remote object by its name ("CalculatorService"), and invokes methods on it.

More info about the server

String[] args in the main Method: In Java, the main method serves as the entry point for any Java application. The String[] args parameter allows you to pass command-line arguments to the program. Although in this example, args is not used, it's a standard part of the method signature. If you wanted to allow the user to specify the port number or other configurations at runtime, you could use args to parse these command-line arguments.

Why Use try and catch? In Java, the try block is used to wrap code that might throw an exception. Exceptions are events that disrupt the normal flow of the program, such as network errors, I/O errors, etc. The catch block is used to handle these exceptions. If an exception occurs within the try block, the code in the catch block is executed. This helps in preventing the program from crashing and allows you to handle the error gracefully.

In the context of Java RMI, many methods can throw RemoteException or other checked exceptions that need to be handled, hence the use of try and catch.

Port 1099 and Its Significance: Port 1099 is the default port number for the RMI registry in Java. The RMI registry is a service that allows clients to look up remote objects by name and obtain references to them. Using a specific port (1099 by default) helps standardize where the RMI registry can be found. However, you can choose a different port if needed, as long as both the server and client know which port to use.

What registry.rebind Does: The registry.rebind(String name, Remote obj) method binds the specified name to the specified remote object in the RMI registry. If an object with the same name is already bound in the registry, it is replaced with the new object. In the example: registry.rebind("CounterService", counter); This line registers the counter remote object with the name "CounterService" in the RMI registry. This name is used by clients to look up the remote object.

How the catch Block Works: The catch block is executed if an exception occurs within the associated try block. It captures the exception and allows you to handle it. In catch (Exception e) e.printStackTrace();, the catch block will catch the exception (of type Exception) and print the stack trace, which provides details about the error.

Compiling and Testing

1. Compile the Java files: javac *.java
2. Start the RMI registry: rmiregistry
3. (Could use rmiregistry (port number) & which runs the process in the background)
4. Run the server: java CalculatorServer
5. Run the client: java CalculatorClient

Actually, after a lot of trying, the CalculatorServer actually includes LocateRegistry.createRegistry(2099). This means that you don't have to manually start the rmiregistry. So I think you can just compile the files, then run java CalculatorServer, then java CalculatorClient.

Port issues

If port 1099 is already in use, run command: lsof -i :1099, to get info. Then run kill (PID) to stop. Can run: ps aux — grep java, to see all java action.